Aspects of Adaptive Reconfiguration in a Scalable Fault Tolerant System *

Stephanie Bryant, Feiyi Wang

Advanced Networking Research
MCNC Research & Development Institute
Research Triangle Park, North Carolina, 27709

Abstract. SITAR is an architecture that incorporates the fundamental ideas of fault tolerant computing such as redundancy, voting, and adaptive reconfiguration, along with acceptance testing in a security setting, in order to enable commercial-off-the-shelf servers to continue providing service in the presence of intrusion or partial compromise. This paper presents the integrated adaptive reconfiguration feature of the SITAR system, which can dynamically change the system configuration in response to the changes in level of intrusion threat in order to maintain service availability. The reconfiguration mechanism has implications on the system design and operation, more specifically the oscillation phenomenon. The discussion covers important aspects of the SITAR adaptive reconfiguration mechanism, including a generic algorithm of its operation, and how it addresses the oscillation problem. In order to actively manage the reconfiguration process and reduce the tendency toward disruptive oscillations, the ARM design incorporates mechanisms according to the feedback loop model and employs several strategies, such as mutual exclusion, steady state monitoring and system observation, for cooperative control. This adaptive mechanism has been implemented and demonstrated as part of the SITAR prototype system.

1 Introduction

Society's increased reliance on software systems has increased the demand for software that operates for extended periods of time, without interruption. The demand for longer operation requires software systems to be flexible enough to adapt in response to environment changes during the longer periods. Moreover, the presence of attacks has become a ubiquitous part of the software system landscape as more and more software systems proliferate to an increasing number of industries. Traditionally, intrusion research software has focused on keeping intruders out [1, 11]. The results of that approach has seen limited success as new attacks render existing intrusion prevention methods ineffective. In response to the challenge of extended operation in an environment containing the potential of intrusion, we concede that systems may not been able

* This work is sponsored by the U.S. Department of Defense Advanced Research Projects Agency (DARPA) under contract N66001-00-C-8057 from the Space and Naval Warfare Systems Center - San Diego (SPAWARSYSCEN). The views, opinions and findings contained in this paper are those of the authors and should not be construed as official DARPA or SPAWARSYSCEN's positions, policy or decision.

to prevent all intrusions. A fault tolerant approach anticipates that some attacks will be successful. However, continued operation in the presence of those attacks enables system to meet the demands of continued operation.

SITAR is an architecture that incorporates the basic ideas of fault tolerant computing [2, 6, 8, 10] such as redundancy, voting, and adaptive reconfiguration, along with acceptance testing. It endeavors to extend fault tolerant capabilities to vulnerable commercial-off-the-shelf servers. Protection provided by the SITAR system enables protected servers to continue providing service in the presence of intrusion or compromise. The integrated adaptive reconfiguration feature of the SITAR system dynamically changes the system configuration in response to the changes in level of intrusion threat in order to maintain service availability. Incorporation of an adaptive reconfiguration mechanism into the SITAR system has implications on system operation. One such implication is an observed phenomenon known as oscillation. For systems with multiple configuration alternatives, oscillation describes a behavior of the adaptation mechanism to repeatedly vacillate among the configuration options. An intruder can potentially use this behavior to inflict an attack on the system, resulting in a denial of service by the system.

The following discussion presents important aspects of the SITAR adaptive reconfiguration mechanism and how it addresses the oscillation problem. Section 2 gives a brief overview of the SITAR architecture. Section 3 explores the SITAR dynamic reconfiguration mechanism and presents a general algorithm of its operation. Section 4 identifies and discusses a recognized side effect of reconfiguration, the oscillation phenomenon; the discussion includes identification of the problem, a potential means of exploiting the vulnerability, and mechanisms used by SITAR and other adaptive systems to mitigate the impact of oscillation. Section 5 includes a discussion of the current adaptive reconfiguration implementation and integration with the rest of the SITAR system as well as details of two demonstration engagements that showcased the SITAR system's functionalities.

2 SITAR Architecture Overview

SITAR (Scalable Intrusion Tolerant Architecture) is a DARPA funded project under the OASIS program [12]. The goal of SITAR is to augment the fault tolerant capabilities of distributed web services, specifically a cluster of distributed COTS (Commercial Off The Shelf) servers. The architecture aims to detect and mitigate the effects of intrusions in order to maintain continued system operation in the presence of intrusion. It provides a framework for implementing a fault tolerant facade in front of a cluster of intrusion vulnerable COTS servers. The architecture incorporates basic ideas of fault tolerant computing such as redundancy, ballot voting, and dynamic reconfiguration, and along with acceptance testing and validation.

The architecture, shown in Figure 1, specifies four key of processing modules. The Proxy Module (PM) accepts client requests on behalf of the SITAR system and returns an appropriate response to the client. The Acceptance Module (AM) uses acceptance testing to check the appropriateness of client requests and validity of server responses. It transmits acceptable requests, via a wrapper module, to a server within the server cluster and retrieves the server response for validation. The Ballot Monitor (BM), using voting

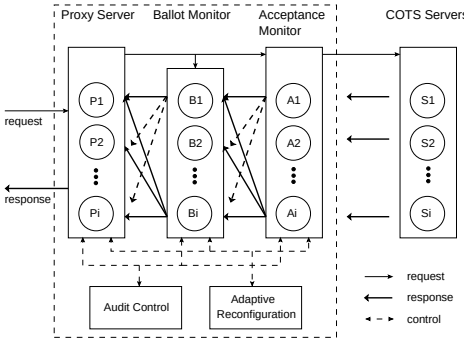

Fig. 1. SITAR Architecture

techniques selects an acceptable server response when there are redundant AM(s), and passes the validated response to the PM(s).

In addition to the processing modules, the SITAR architecture contains two monitoring modules, AuditControlModule (ACM) and the AdaptiveReconfigurationModule (ARM). The ACM, using distributed agents, monitors execution parameters of SITAR processing modules and COTS servers. In addition, it performs trending analysis to predict when COTS server execution degrades beyond acceptable limits and employs software rejuvenation techniques to persuade the server to resume acceptable operation. The ARM acts a system supervisor [14] that monitors the SITAR processing modules operation. It is responsible for maintaining a global system snapshot of the distributed components participating in processing, analyzing trigger messages to determine the level of threat perceived by the system and the probable source of system compromise, and changing system components to ensure continued operation as the execution environment changes.

The SITAR system can be configured into one of many user specified postures, at any given time, in order to mitigate the effects of malicious attacks. As the system detects the effects of attacks to the SITAR modules and/or the COTS servers that it protects, the ARM dynamically changes systems threat posture configuration to increase redundancy of checking, voting, and the number of concurrent request sent to different COTS servers in the cluster. The current SITAR implementation uses three (3) threat postures configurations, which the ARM chooses among, based on the perceived system threat. When SITAR system is under low threat from the environment, the SITAR system is configured with minimal checking for faster performance. While at higher levels of threat, the system increases redundancy for increased checking and continued availability, with albeit degraded performance.

3 SITAR Adaptive Reconfiguration

3.1 Design Analysis

In the SITAR architecture, the ARM is primarily responsible for planning and initiating configuration changes in the system. The ARM is capable of changing attributes

of SITAR modules or selected aspects of the user policy. It has access to information generated by the SITAR modules and user specified policy information. In order to develop a suitable reconfiguration mechanism, it is necessary to examine the nature of the available information and how it describes the system operation in order to determine what changes the ARM can make to the SITAR modules to affect request/response processing and system availability.

In order to assess the set of system information available to the ARM, the overall system architectural specification and the module behavior are examined. According the architecture specification shown in Figure 1, all modules send trigger messages when they detect compromise or errant behavior in other SITAR modules. In addition to trigger information, the modules also transmits a wide variety of state information to the ARM. The state information includes static and dynamic parameters. Static information includes identification information, e.g. name and module type, as well as processing capability limits such as the maximum number of concurrent processing paths that it can participate. Dynamic information, configured at initialization and changed during execution, encompasses parameters that track the modules current activity, for example the number of computing resources that are busy processing request/response exchanges, and what service protocols the modules understands, such as HTTP and DNS. Other information that may be useful for reasoning includes performance statistics, but the modules do not currently transmit that information. Together module's static and dynamic information construct a profile of the system's limits and its current behavior [15], as well as its availability to assume additional work. Refer to textbox below for examples of module state information. User specified policy information is made available to the ARM. The policy information details the acceptable configurations along with their operational parameters. The policy information describes when a specific threat posture is applicable and the thresholds of operating within the posture. When the system exceeds the posture threshold, a new threat posture needs to be selected and the system should be changed to comply with the new threat posture configuration.

```
              Sample Module State Information

// Static Module Information:
public String name;                //Unique module identifier
public String componentType;       //SITAR module type
public String hostName;            //Host machine name
public Integer maximumWorkers;     //Max. #  resources
......

// Dynamic Module Information:
public String[] serviceTypes;      //Types of protocols
public String configStatus         //Module operational status
public Integer currentWorkerLimit; //Max. # available resource
public Integer busyWorkers;        //# working resources
public Long reconfigAckId;         //Reconfiguration directive
                                   //  acknowledge identifier
......
```

After examination of the system behavior, the set of changes that the ARM can make to each SITAR module are identified. Knowing a module's maximum resource capability and its current resource usage, the ARM can increase or decrease the number resources available of processing. Resource capability adjustments enable the processing capacity to vary with a given threat posture. For example in times of heavy workload, more resources can enabled to system to increase request load while not changing threat level, which adds more checking. When a SITAR module experiences compromise, the entire module may be compromised or a service that it provides may experience problems. The ARM can exclude an entire module or a service within a module. Exclusion here means that either the module is prevented from accepting work for a specific service or for all services. In the event that a module is prevented from accepting work for all services, it is prevented from accessing the main communication infrastructure, isolating it from the system. SITAR modules have three general operating states - configured, un-configured or excluded. These states describe a module's participation in request/response exchange processing. Configured modules actively participate in processing, i.e. perform their assigned checking functions; Un-configured modules perform minimal tasks to maintain group membership, but do not participate in exchange processing; Excluded modules are prevented from communicating with other modules in the system. The ARM can change the operational state of a module to affect system availability, by configuring more modules, freeing unneeded modules, or removing compromised modules.

As the number of modules increases, the ARM can become overwhelmed with information. As a means of managing the quantity of information received by the ARM, it employs separate message collection paths to gather and perform interim analysis on incoming information before the reconfiguration mechanism is notified. The ARM uses trigger information to reason about the hostile nature of operating environment, expressing it as a numeric quantity for comparison with established thresholds, and identify which modules have been compromised and must be excluded from the system. The state information is used to identify resource inadequacies or excess when compared to the user defined threat posture configurations.

Development of this adaptive reconfiguration mechanism requires identification of output information provided by the system and input information that the mechanism can inject into the system to affect change. The SITAR system provides trigger messages and module state information in order for the mechanism to access the current system behavior and the degree of threat present in the system. The mechanism changes the SITAR system by transmitting directive orders to alter the SITAR modules. Those changes include changing aspects of the modules internal state such as as its operation state, the number of resources (within its specified limits), and the service offerings. In addition to system generated information, the mechanism also has access to user specified policy parameters that define the scope of the system configuration with accompanying thresholds. The manner in which information is exchanged between the ARM and the SITAR modules is consistent with the feedback loop module, meeting many of the requirements identified by Shaw [13] that enable one to apply this model to the SITAR system.

3.2 SITAR ARM Algorithm

The adaptation mechanism in SITAR system is modeled as a feedback loop system [13]. The mechanism continuously evaluates the available system information and changes the system to ensure continued system availability as the system execution parameters and environment change. The following presents the input, the output, and the general steps of the reconfiguration mechanism used to manage the SITAR system.

A. Input

- **Policy (P)** User preferences are input into the ARM through a configuration file that describes the governing policy of the system operation. The policy contains descriptions of each threat posture configuration along with its associated threshold parameters. The threat configuration indicates the minimal number of modules of each module type that are required in order to provide adequate checking and validation of request/response exchanges for a specific threat posture. The threshold parameters are numerical limits that establish the bounds of the configuration applicability, such as acceptable thresholds for trigger message rates and perceived environment threat rating. When the system operation exceeds the thresholds, a new threat posture is chosen.
- **Reconfiguration Condition (C)** The reconfiguration condition describes a persistent situation that may warrant reconfiguration action. Resulting from analysis of either state or trigger information, the condition represents diagnosis of a problem or a reason to change the system and may be optionally accompanied by additional information describing the extent of the discrepancy. For example, state messages could reveal that not enough modules are in the configured state for the current threat posture and the additional information indicates the numeric extent of the deficiency.

  One reconfiguration condition may occur as the result of the values of one or more system parameters and required changes to rectify the condition that may affect more than one other system parameters. So a condition must also include other information to help the reconfiguration mechanism determine appropriate actions. In order to generalize the reconfiguration operation, the parameters and comparison strategies for each reconfiguration condition can be defined separately, e.g. in separate reusable classes, in an object oriented system, and grouped as needed to support the condition. The supporting condition information includes:

  - **Snapshot Parameters** a list of parameters in the snapshot that provides information about this condition.
  - **Policy Parameter Mapping** a list of policy parameters that can be used to assess the extent of the condition.
  - **Comparison Strategy** definition of how to compare the snapshot and policy parameters to confirm the existence of the condition.
  - **Change List** an ordered list of related change actions that may rectify the condition. For example, excluding a module may cause a resource deficiency so a related change would be to try configuring a stand by module of the same type.

After the first change is applied, changes are applied in order to attempt to rectify implications of the change. The mechanism attempts apply the minimum number of changes to rectify the conditions, so all changes by not be necessary given the system current state.

– **Snapshot (S)** The system snapshot is a special input. It is an active repository that represents the latest state information of the module participating the system. SITAR modules are distributed throughout the network, so the snapshot is central repository of the states of the all the modules. The snapshot is derived from the state update information sent by the modules and is consulted by the reconfiguration mechanism to confirm the persistence of conditions and provide a base for reconfiguration changes. The snapshot is described as active because, it can also generate reconfiguration conditions if incoming state messages cause conflicts between the operating system state and the configuration requirements contained in the policy.

B. Output

The reconfiguration mechanism produces changes directives. A change directive describes one aspect of the system that should change. The mechanism effects changes to the policy and/or system. Below are descriptions of each class of output change directives. Since the reconfiguration analysis process may be an involved computation, the ARM attempts to deliver as many related directives in one shot, so the actual output of the reconfiguration mechanism is a set of change directives, the output directive, denoted as D in the algorithm, may consist of one or more change directives, where each change directive has a specified destination - the global policy or a specific module. For directive issued to modules, the ARM keeps track of the directives, awaiting acknowledgment from the target module that the change has been completed. Failure to complete the reconfiguration directive is viewed as not compliance which may result in additional action.

– **Policy Change** The ARM is permitted to update the policy information by setting the threat posture selector to choose a new threat posture. In the current SITAR implementation, the system can be configured in one of three threat posture configurations, labeled 0, 1, and 2. If analysis of trigger information reveals the threat posture thresholds have been exceeded, the the ARM may choose the next higher threat posture by changing the threat posture selector presently at 1 to 2.
– **System Change** The system that the reconfiguration mechanism endeavors to control is the set of SITAR processing modules. So a system state change is a change to the internal state of a SITAR module ordered by the reconfiguration mechanism. Changes include adjusting a module's resource capabilities (within its limits), changing its operational state, and/or excluding one of its services or the whole module from the system.

C. Algorithm Definition

Algorithm 1 describes general procedural steps performed by the reconfiguration controller to determine what reconfiguration is warranted, using the system snapshot

and the global policy as input. The algorithm assumes that the module state update time is reasonably faster than the processing time so that the ARM can maintain a reasonable representation of system activities.

Algorithm 1 : Reconfiguration Algorithm

Require: global policy, P
Require: system snapshot, S
Require: reconfiguration condition, C
Require: boolean indicating 1 or more snapshot parameters violate policy requirements, $violationExists$
Require: boolean indicating reconfiguration is necessary, $reconfigNeeded$

```
 1: tl ← retrieve current threat posture value from P
 2: pparams ← retrieve the values of policy parameters associated with C
 3: sparams ← retrieve the values of snapshot parameters associated with C
 4: reconfigNeeded ← results of C's comparison strategy of pparams and sparams
 5: if reconfigNeeded == TRUE then
 6:    while C's change list has more entries AND violationExists == TRUE do
 7:       retrieve the next element in C's change list
 8:       compute appropriate amount(s) to change parameter(s) to satisfy C
 9:       select a module to reconfigure
10:       create new changeAction with parameters module ID, parameters, changeAmounts
11:       if changeAction == PENDING then
12:          discard changeAction {prevent re-issue of duplicate changeAction awaiting ac-
             knowledge}
13:          continue to next iteration of while loop
14:       else
15:          add changeAction to D
16:          generate a new snapshot, S', by adding changeAmount to S
17:          violationExists ← reconcile S' with P
18:       end if
19:    end while
20:    Output D
21: end if
```

The algorithm starts by retrieving the current threat posture value from the global policy which determines the specific threat posture and threshold values that applies at the current time. The appropriate values of the global policy and the snapshot parameters associated with the condition are retrieved and stored in local variables. The comparison strategy associated with condition C is used to compare the snapshot and policy values, determining if reconfiguration is warranted. If reconfiguration is warranted, then the algorithm attempts to satisfy the condition, by making changes to snapshot parameters directly affected by the condition. A new snapshot is computed as each parameter is changed. The new proposed snapshot is compared with the policy values to see if any of its parameters violate the thresholds. If suitable changes can be found, the change directives are sent which change the system state values accordingly with acknowledgments sent when the change is completed, as necessary. The cycle ends when all the changes

in the change list have been attempted to derive an acceptable snapshot. If at the end of the cycle a violation still exists, a new cycle is may be initiated which incorporates new module state updates. The goal of the algorithm is make reasonable changes based on the policy. However, it may require more than one cycle of examination in order to resolve all conflicts.

D. Algorithm Implementation

The SITAR ARM implements the reconfiguration algorithm, described in the previously, which adjusts SITAR system parameters based on analysis of various aspects of the system operation. The mechanism employs are two loosely coupled feedback paths, shown in Figure 2, which provide immediate as well as longer term responses to system conditions. The inner feedback loop, containing the `SystemSnapshot` component, incorporates state update information into a global view of the modules participating the system. This loop provides fast response to reconcile isolated resource issues. For example, when a module joins or leaves the system, the resource and service availability of the system changes. The changes may require reconfiguration action to the configure, i.e. change to an active state, one or more SITAR modules, or advertise support for a new service. Also, if the system is experiencing a high workload that is not related to an attack, modules may be directed to increase their worker resources to accommodate the extra load. The outer feedback loop performs more analysis and attempts to reason about interdependencies among the system modules and about the overall state of the execution environment, particularly the perceived threat from the environment. This analysis may take longer, requiring more analysis cycles to arrive at a reasonable conclusion. The results of that analysis may cause more substantial changes that may alter the global policy and change internal states of multiple modules. The reconfiguration actions of this loop are intended to affect the system availability over extended time periods.

In Figure 2, the `PolicyRules` component collects user configurable parameters that are set before system initialization and can be changed during system execution. The `PolicyRules` component generates the global policy that governs the reconfiguration operation. The global policy is accessible by all mechanism components that require policy input, so in the figure all references to “Global Policy” refer to the policy generated by the `PolicyRules` component. The core of the reconfiguration mechanism is the `ReconfigurationDirector`, which accepts as input the global policy as well as environmental information and generates a variety of reconfiguration directives to change the state of the processing environment and/or update the global policy.

In this feedback system model, the environment to be controlled is defined as the set of SITAR modules processing the request/response exchanges. During execution, modules send two types of information, state update and trigger messages, to the `ARM`, which uses this information to reason about the SITAR system operation and the execution environment. The state update messages provide update-to-date information about modules’ resource capabilities. Modules send trigger information when they detect anomalous behavior in other SITAR modules or COTS servers. The state update and trigger information are analyzed along separate feedback paths.

Periodic state update information from the modules enable the `ARM` to develop and maintain a global view of the modules participating the SITAR system; the view is

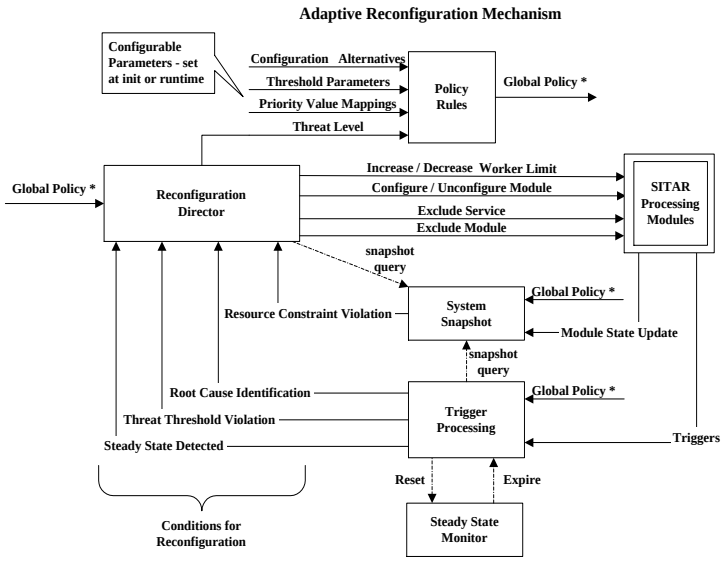

Fig. 2. Reconfiguration Feedback Loop Mechanism

called the system snapshot. The `SystemSnapshot` component actively updates its information based on the information contained the module state update messages. It is the central repository that facilitates reasoning about the SITAR environment, as the modules are distributed throughout the network residing on any number of independent host machines. The `SystemSnapshot` accepts queries from other mechanism components, providing them with pertinent information for their reasoning and analysis. As state update information is incorporated into the `SystemSnapshot`, analysis is performed to determine whether the new system state meets the system requirements set forth in the global policy. Discrepancies between the system's state and the system requirements are identified as sufficient reasons to perform reconfiguration. Analysis of the `SystemSnapshot` information generates resource violation messages that are sent to the `ReconfigurationDirector`. Resource violations are defined in terms of discrepancies in processing resources, such as there are not enough configured modules or processing workers for the current threat posture configuration.

Trigger messages are sent to the `ARM` by other SITAR modules when they detect anomalous behavior in their neighbor modules. The errant behavior or compromise may be an effect of a malicious attack. Thus, analysis of trigger messages allows the system to assess the level of threat, malicious or unintentional, present in the environment. The first phase of trigger processing performs fast event correlation in order to determine the source of the compromise. The event correlation uses a two step process to isolate a module or service that is compromised. The first correlation uses a sliding time window to analyze the triggers as a group, preventing unnecessary action for transient trigger occurrences. The triggers are categorized based on information that they contain, e.g.

the suspected source of the problem. If any category of triggers exceeds the user established policy threshold, an alarm condition is declared. An alarm condition represents a persistent problem in the system. The second stage analyzes the alarms to determine if action is warranted. The alarm analysis generates histogram of alarm information for each service and examines it to identify which module has proportionally more triggers identifying it as the source of the compromise. If a module or service is identified as the source of compromise, action is taken against it. The second phase of trigger processing uses the number or occurrences a trigger and its priority classification to calculate a number rating of the perceived system threat. The threat value is the sum of the products of the number of occurrences and the numerical priority value for each class of triggers. If the calculated threat value exceeds the established policy threshold, then reconfiguration action is warranted. While triggers indicate increase environment threat, lack of triggers indicates no new or diminishing threat and in response, the system can assume a less suspicious posture. The mechanism employs a steady state monitor to recognize periods where no new triggers are received. When the steady state period expires, then system may be changed to configuration with decreased redundancy and higher performance. The trigger processing and steady state monitoring operate in an exclusive manner with the trigger processing receiving precedence. So when a trigger is received, the steady state monitoring is halted and is only resume after sufficient time has lapsed without any triggers.

The reconfiguration conditions identified during the analysis of the environmental information is sent to the core reconfiguration agent, the `ReconfigurationDirector`, which determines what changes to make the system. The `ReconfigurationDirector` determines what actions to take based on the condition that is identified during analysis and information in the current snapshot. The reconfiguration directives change the states of the SITAR processing modules and/or change the policy's threat posture value. The policy threat posture value determines which threat posture configuration that the system assumes and which thresholds to consider when making reconfiguration decisions.

4 Oscillation Problem and Solutions

4.1 Problem Definition

In systems with multiple configuration alternatives, the tendency to continuously fluctuate between configuration options is the phenomenon referred to as oscillation. As an adaptive system accesses its environment, it makes changes to the target system's configuration in response. However, if the changes are too frequent and uncontrolled, the system may never converge to one of the accepted configurations, resulting in unintended system instability. While eliminating the ability of the system to change according to the environment prevents oscillation, it also relegates systems to the statically configured system that the industry is attempting to evolve from. An alternative is to minimize the impact of the systems oscillation tendency by enacting control measures for the adaptation mechanism.

Other research projects acknowledge the existence of the problem and some include methods for mitigating adaptation oscillation. Chen [4] describes an adaptive system

that includes adaptation aware modules (AAM) at various levels of the system architecture. The discussion suggests the development of a selection function which "must also prevent oscillation between alternative AAMs". Karsai [14] recommends avoiding "situations where the system spends most of its time switching between configurations" and acknowledges that further study is needed to define stability for self adaptive software. In the Ginga system developed by Paques et al., the adaptive query processing system uses a wait time to "prevent oscillation in feedback-based adaptation" [9]. The use of such a wait affects the speed of reconfiguration, possibly introducing delays if the time too long. Pagques points out that development of a precise wait time is a hard problem requiring predictions of the environment's future behavior. Managing conflicting reconfiguration goals, the Willow project [3] employs a resource manager and priority enforcer, ANDREA, to issue reconfiguration changes in partial ordered fashion according to the established priorities of reconfiguration actions. ANDREA is preemptive giving higher priority reconfiguration changes precedence and uses transactions to allow rollback of failed changes.

In the fault tolerant systems such as SITAR, any component can be the target of attacks, including the system supervisor, ARM. While the ARM design does include the capability of redundancy in case of primary ARM failure, an attacker could employ a manipulative stream of requests that attempts to trigger repeated system reconfigurations in short spans of time. The rapid reconfigurations would become an internal system disruption, that may severely impact system performance or availability. This process can be viewed as a form of denial of service attack exploiting the oscillation problem. The next section discusses how the SITAR attempts to mitigate such an attack.

4.2 SITAR Mitigation Mechanisms

The SITAR adaptive reconfiguration mechanism changes the system to one of a known set of configuration alternatives or threat postures. The mechanism applies the configuration alternatives in order where lower threat postures provide higher performance with reduced checking and higher threat postures provide increased checking through redundancy and continued availability in the presence of intrusions. The goal of the adaptation mechanism is to response quickly to intrusions increasing redundancy for checking and availability, however, the quick response should be tempered so that the reconfigurations do not impede the system operation. Figure 3 depicts the desired adaptation response to environmental hostility represented by intrusion trigger activity. The graph demonstrates that the system should responds reasonably quickly as the environment hostility increases. However, the adaption behavior does not precisely follow the trigger behavior and move to lower threat posture when it first detects a significant decrease in triggers. The goal of a delayed descent to lower threat level, enables the system maintain an adequate threat posture if the threat resumes after a short period of time. In the graph, when the barrage of trigger messages resumes, the system is already in an elevated threat posture, so system is in position to respond to the continued attack and mitigate its effect, without additional reconfiguration.

The SITAR adaption mechanism discussed in Section 3 implements this desired behavior. Referring to Figure 2, the initial detection response is handled by the trigger processing feedback loop, when an escalating rate of triggers is detected, the trigger

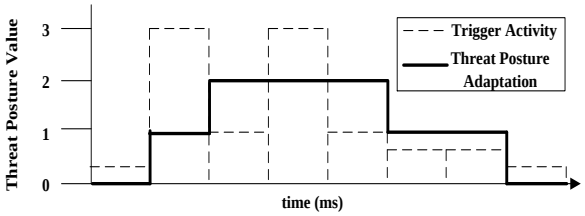

Fig. 3. Desired Posture Adaptation Behavior

processing determines that the system should increase its threat posture. In order to control the ascent to different threat posture, the mechanism changes the threat posture to the next highest threat posture one at time as needed until one posture is judged suitable or the highest posture with the most redundancy is reached. Increasing the threat posture is given precedence, as it increases redundancy and availability

In opposition to increasing the threat posture, the `SteadystateMonitor` tries to detect the absence of triggers so that the system can move to a lower threat posture increasing performance. These two opposite system drives affects the mechanism's tendency to oscillate among the threat posture alternatives. It is here that the SITAR system implements control so that the competing drive to descend to lower threat posture for performance is balanced with the systems desire to meet its fault tolerance goals. First, the two opposing drives are linked allowing them to operate in mutually exclusive manner. That is, the trigger processing which increases the threat level can preempt the steady state monitor so that the system can response to escalating threats promptly. The trigger processor restarts the monitor when trigger messages cease. Second, the `SteadyStateMonitor` employs at timer to monitor the system for an absence of triggers over a period of time, the steady state period. The period[1] for steady state monitoring is user configurable and introduces a delay just in case an attack should resume; the system remains at a sufficient posture to mitigate the effects the renewed attack. When the steady state period has elapsed with no trigger messages, the mechanism does not blindly descend to a lower threat posture because modules may be still working on requests that arrived during the attack. Moving to a lower threat posture may significantly affect the system's processing capabilities by removing needed resources, thus disrupting the processing still in progress. Once the steady state is reached, the adaptation mechanism checks the system resource usage. The mechanism will not attempt to reduce the threat posture until the system resource usage begins decreasing as some processing completes. This slower descent allows the system to complete processing, release resources in an orderly fashion, and the adaption mechanism to move the system lower threat posture in a controlled manner that reduces the tendency toward oscillation. The SITAR adaptation mechanism mitigates the effects of oscillation in re-

[1] As mentioned above, the use of a static time period has some drawbacks. An alternative approach could be to use a time period that is determined by a probabilistic distribution around a mean time, producing some variability and increasing the work factor for the attacker attempting to tailor a request stream.

sponse to substantial environment fluctuations by implementing mechanisms to control both the ascent and descent between threat postures; the mechanism employs mutual exclusion of competing drives, a steady state period, as well as incorporating current system activity in a manner that discourages oscillation among the threat level postures.

5 System Prototype and Demonstration

The adaptive reconfiguration mechanism described in the previous sections is implemented in the adaptive reconfiguration module (ARM) in the SITAR prototype system. The SITAR prototype, including the ARM, is implementing in Java Programming Language and uses Sun Microsystem's Jini [7] and JavaSpace [5] technology as its main communication infrastructure. The ARM employs modular, multi threading, and synchronization design techniques in order to implement a loosely coupled, yet cooperative, set of subsystems to realized the adaptive reconfiguration functionalities. Following system initialization, the ARM advertises, to active SITAR proxy modules, the system's service and resource availability based on the state information provided by registered modules. The Proxy Module establishes module participation in processing pipelines through request and response negotiations. Subsequently, the ARM changes the service offerings and resource availability as the module's changes state, through client-server request/exchange processing, and levels of participation in the system. In addition, the ARM reasons about the environment using the state and trigger information, transmitting appropriate reconfiguration directives the adjust the resource availability or change the system threat posture.

The prototype implementation was publicly demonstrated on two occasions presenting various aspects of the SITAR functionality. The first demonstration at the DARPA OASIS 2002 Summer PI Meeting Hilton Head South Carolina included the first stable implementation of the adaptive reconfiguration mechanism. Although it used one static configuration, the presentation demonstrated the overall flow of control information between the ARM and processing modules. At that stage, the ARM was able to retrieve the available system information, maintain a system snapshot and preform cursory checking of the state and trigger information. The system configuration space consisted of one base threat posture configuration. At that stage of development, the system demonstrated its ability to protect a server cluster consisting of three web servers. The demonstrations showed SITAR's ability to detect and mask, through acceptance testing and ballot voting, the effects of a simulated Code Red attack while continuing to deliver appropriate responses to the client.

In the second demonstration at the 2003 DISCEX III Conference in Crystal City Virgina, the SITAR prototype implementation contained a more extensive adaptive reconfiguration implementation that performed the system adaptation as a described above. The ARM managed a system snapshot, performed reasoning about the system information and dynamically changed the system threat posture accordingly. The implementation included three threat posture configurations and provided service for DNS and HTTP requests. The ARM dynamically adjusted the service and resource availability advertisements as modules entered, left, or were removed from the system. In addition, the system changed the system among the threat postures in response to the cues

from the environment, i.e. triggers from SITAR modules. The demonstration showed that the system could detect and tolerate the effects of attacks to the COTS servers and SITAR modules. The attack scenarios included a variety of direct and indirect threat situations. The client initiated an attack by submitting a malicious request and SITAR prevented the propagation of errant response from compromised server. A covert attack compromised one of the servers and SITAR detected the difference in server response, prevented the dissemination of corrupted information. Through fault injection, a ill behaved SITAR module was allowed to participate in the system. Once detected through trigger information, the ARM removed the misbehaved module and configured a standby module so that the system could continue operation and raised the system's threat posture, if warranted. SITAR was also able to detect the effects of a memory leak attack, exploiting a server vulnerability were memory was not being completely freed after a simple request. SITAR detected the abnormally high rate of memory usage by the server through predictive trending and determined when to restart the server before it experienced total memory consumption and subsequent failure. In addition to direct attacks, SITAR also was able to withstand natural faults due to physical network separation. In that case, the network cable connecting a module's host machine to the network was removed and the module was not physically able to perform its membership maintenance duties. The ARM detected the modules absence through missed keep alive messages, removed the module's information from the snapshot and configured another stand by module so that the system could continue operating.

6 Conclusion

Industries are embracing software systems that operate for extended periods of time, adjusting to environmental changes that may include attacks. SITAR is a intrusion tolerant architecture that incorporates the basic of fault tolerant computing such as redundancy, voting, adaptive configuration, along with acceptance testing. The goal of SITAR project is to augment fault tolerant capabilities of vulnerable COTS servers, enabling them to continue providing service in the presence of attacks. Implementation of the architecture detects and masks the effects of attacks through dynamic redundancy adaptation, acceptance testing and ballot voting. SITAR's integrated dynamic reconfiguration mechanism uses a feedback loop model to change the SITAR configuration and resource parameters in order to sustain continued system availability. Examination of the SITAR architectural specification and operation used to develop the reconfiguration mechanism was discussed. The resulting general reconfiguration algorithm was presented. Systems, such as SITAR, with multiple configuration alternatives can experience the oscillation phenomenon, where the system continuously wavers between configuration alternatives. SITAR uses three complementary strategies to minimize the impacts of oscillation. It uses a steady state monitor, mutual exclusion of drives to increase and decrease the system threat posture, and system operation inspection to control the changes in configurations, so that system reconfigurations are performed in a controlled manner in spite of potentially erratic environment fluctuations. The adaptive reconfiguration mechanisms as describe in implemented in a SITAR prototype and was demonstrated on at the DARPA OASIS 2002 Summer PI Meeting and the 2003

DISCEX III Conference. Each demonstration showcase the various capabilities of the SITAR system, including the adaptive reconfiguration mechanism for monitoring the SITAR modules and changing the system's threat posture in response to environmental conditions.

References

1. E. Amoroso. *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace back, Traps, and Response*. Intrusion.Net Books, 1st edition, January 1999.
2. J.-C. Laprie Avizienis and B. Randell. Fundamental concepts of dependability. Technical report, LAAS-CNRS, April 2001. Research Report N01145.
3. John C. Knight Dennis Heimbigner Alexander Wolf Antonio Carzaniga Jonathan Hill Premkumar Devanbu and Michael Gertz. The willow architecture: Comprehensive survivability for large-scale distributed applications intrusion tolerance workshop. In *DSN-2002 The International Conference on Dependable Systems and Networks*, Washington DC, June 2002.
4. Wen-Ke Chen Matti A. Hiltunen and Richard D. Schlichting. Constructing adaptive software in distributed systems. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, pages 635,643, Mesa, AZ, April 2001.
5. Sun Microsystems Inc. Javaspaces^{TM} Service Specification. `http://www.sun.com/jini/specs/js1_1.pdf`.
6. P. Jalote. *Fault Tolerance in Distributed Systems*. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1994.
7. K. Arnold, editor. *The Jini*$^{\mathrm{TM}}$ *Specifications*. Addison-Wesley, 2nd edition, 2000.
8. P. A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice*. Springer Verlag, 1990.
9. H. Paques L. Liu and C. Pu. Ginga: A self-adaptive query processing system. In *Proceedings of ACM-CIKM*, 2002.
10. M. R. Lyu. *Software Fault Tolerance*. John Wiley Sons Ltd, 1995.
11. S. Northcutt. *Network Intrusion Detection: An Analyst s Handbook*. New Riders, 2001.
12. F. Wang (PI) K. Trivedi (Co-PI) R. Wang S. Bryant R. Uppalli C. Killian B. Madan K. Vaidyanathan R. Poonamalli and D. Chen. SITAR: Scalable intrusion tolerant architecture for distributed services. Technical report, MCNC Research & Development Institute, April 2002.
13. Mary Shaw. Beyond objects: A software design paradigm based on process control. *ACM Software Engineering Notes*, 20(1), January 1995.
14. Karsai G. Ledeczi A. Sztipanovits J. Peceli G. Simon G.and Kovacshazy T. An approach to self-adaptive software based on supervisory control. In *IWSAS-2001 (submitted)*, Balatonfured, Hungary, May 2001.
15. D. Rosu S. Yalamanchili and R. Jha. On adaptive resource allocation for complex real-time applications. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS)*, San Francisco, USA, 1997.